Supporting Collaboration between Customers and Developers:

Supporting Collaboration Between Customers and Developers: a Framework for Distributed, Agile Software Development

Francesco Bergadano, Gianni Bosio, Stefano Spagnolo
Dipartimento di Informatica, Università di Torino, Italy

ABSTRACT
The traditional, in-house software development process is progressively losing its appeal in favor of distributed, multi-site development: this is mainly due to the well-known advantages of the latter, such as higher productivity and lower costs. However, this practice has also some documented disadvantages that are inherent to distance: geographical, temporal and socio-cultural distances between stakeholders can affect communication, coordination and control activities, thus making collaboration very difficult. This would not immediately look as the ideal scenario for applying Agile methodologies, which definitely rely on continuous collaboration between all stakeholders, including (with a very important role) the customers. This paper analyzes issues related to collaboration between customers and developers in a distributed, Agile setting and proposes a framework that defines practices and tools for handling project information and communication activities.

Keywords: Customer - Developer Collaboration, Distributed Software Development, Collaborative working environments.

INTRODUCTION
A great part of the activities that happen in software development projects are usually enacted between people that are responsible of the development phase of the project itself: developers, analysts, testers, managers. However, a project features other stakeholders that can be involved in collaborative activities, covering one or more phases of the project, such as customers and final users.

Collaboration between the development team and customers can have several positive effects: in fact, correct requirement analysis can lead to a better comprehension of the customers' real needs, in order to develop a final product which is better responding to such needs and, therefore, fully appreciated (Tseng & Du, 1998). Collaborating with customers can, especially during the early phases of a project, foster an effective exchange of information, which can lead to identifying and correcting errors and defects in the product (Saiedian & Dale, 2000).

Agile development methodologies propose a different paradigm that regulates interaction between customers and developers: specifically, they acknowledge the importance of a continuous flow of communication between them; such flow should comprehend the full duration of the project and should not be limited to the early phases. However, the maintenance of a communication flow can be difficult when, as it happens in Distributed Software Development, different people can be physically located very far from each other: while this is a condition that affects all kinds of distributed development projects, it is particularly severe in Agile ones, as they strongly rely on informal communication and continuous collaboration.

This paper presents the SCoAP (Support for Communication and Agile Project management) framework, a set of collaborative practices and tools that addresses communication, coordination and control issues between customers and developers in Distributed, Agile Software Development projects. Our target is to facilitate situated and contextual communication between developers and customers across remote sites.

The framework is composed of the following three layers:

- *User Definition Layer*: an informative space which collects and presents information about users, allowing to track their personal information, role and current activities.
- *Project layer*: practices and tools which facilitate awareness regarding project and tasks information and enables stakeholders to exploit it as a context for coordination and control strategies.
- *Communication layer*: practices and tools which leverage both formal and informal communication processes in order to promote the sense of proximity between stakeholders.

The paper extends our previous work on the subject (Bergadano & Bosio, 2013) by describing in detail our prototype implementation of SCoAP.

DISTRIBUTED SOFTWARE DEVELOPMENT

In *Distributed Software Development* (DSD), also known as *Global Software Development* (GSD), the activity of developing the same software product is scattered between geographically distributed locations (Prikladnicki et al., 2003; Damian & Moitra, 2006; Carmel, 1999). The industry has been pushed to transform software development into a "multisite, multicultural, globally distributed undertaking" (Herbsleb, & Moitra, 2001) by several factors:

- Cost reduction is possible, if activities are outsourced to countries whose labor is cheaper.
- It can be possible to reach skilled workers and involve them in a project, no matter how distant they are.
- People with different backgrounds can share their ideas and solutions, thus promoting innovative problem solving.

Considering these factors, it is easy to understand why DSD has reached such a considerable level of diffusion. However, along with its advantages, DSD also poses several problems and constraints, that can hamper the final result in various forms (Ågerfalk, 2005; Conchuir, 2006):

- Long travels are needed in order to compensate for the obvious fewer opportunities for in person, face-to-face interaction, which is fundamental for building trust (*Geographical* distance).
- Time zone differences cause different time shifting work patterns, therefore reducing the opportunities for synchronous collaboration (*Temporal* distance).
- Languages, values and normative practices can be dramatically different in long distant countries, making difficult for people to comprehend each other and possibly generating miscommunication issues (*Socio-cultural* distance).

A straightforward solution to such issues is reducing distances; this is the case for another form of DSD, where contractors are located at a shorter distance from their customers, known as *Nearshoring* (Carmel & Abbott, 2007). In such cases, proximity mitigates difficulties related to temporal, geographical and socio-cultural distances, while keeping (almost) the same benefits in terms of cost reduction and product quality. Although Nearshoring is often seen as a preferable choice (Carmel & Abbott, 2006), it still inherits several hurdles and risks that are typical of any DSD process (Carmel & Agarwal, 2001):

- *Communication* is a fundamental element in software development. Developers spend much of their time engaging in both formal and informal communication activities (Herbsleb & Mockus, 2003). However, studies (Allen & Fustfeld, 1975) show that frequency of communication drops off when coworkers are separated, even if two offices are just 30 meters apart. Change management can be seriously affected, as changes need to be

propagated and negotiated very quickly between sites, possibly exploiting informal, lateral communication that is not possible in a distributed scenario.

- *Coordination* issues arise: if communication patterns are not clear and efficient, wrong assumptions can be made about different sites, whose activities can be misinterpreted (Herbsleb & Mockus, 2003). Work issues take significantly longer times to be resolved, when more sites have to coordinate themselves in assigning the most suited people and resources to a specific activity or to a common goal.
- *Control* over the development process and its tasks is harder if there is a lack of information sharing about the activities of the team. This can be a serious problem between coworkers and also between developers and customers, who can have the legitimate desire of monitoring the evolvement of the project and its adherence to their goals and expected quality levels (Herbsleb, & Moitra, 2001). Moreover, communication and common understanding of stakeholders belonging to different communities should be supported by providing them with opportunities to construct their own work environment and have control in the description of problems (Zhu et al., 2011).

AGILE PRINCIPLES FOR CUSTOMER-DEVELOPER COLLABORATION

The traditional development process, also known as *waterfall* development, prescribes customers' direct intervention in the first phase only, i.e. requirement analysis: analysts interview the customers, then they compile a formal document which defines the requirements of the software product; this document must be accepted by the customer and it is the primary element that will determine the route of the development project for all its lifetime (Boehm, 1988). Once development activity ends, customers are once again involved in order to evaluate the resulting software and its adherence to their requirements. As it emerges from this brief analysis, the relationship between customers and the development team are substantially formal and bonded by what has been agreed during contractual negotiations.

A consequence of this approach is the limited influence that customers can have during the actual development process once it starts. The development activity is completely in the hands of programmers, with the possibility that errors or misunderstandings persist until when it becomes very onerous to correct them: it is only possible to spot such errors through the scheduling of possibly onerous formal meetings with the customers.

When using Agile development methodologies, communication between customers and developers comprehends formal meetings and minimal documentation, but it is mostly composed of informal activities. In fact, they claim the importance of keeping a continuous flow of communication with someone that represents customer's interests in the project but that is also aware of the issues of the development process (Paetsch et al., 2003); the eXtreme Programming (XP) proposal extend this concept to the point that a customer's representative has to be physically co-located with the development team (Beck & Andres, 2004; Glass, 2001).

The Agile Manifesto (Fowler & Highsmith 2001) identifies twelve fundamental principles as the basis of every Agile methodology. We can count four of these principles that explicitly mention, or are closely related to collaboration issues between customers and developers:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software* (Principle 1).
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage* (Principle 2).
- *Business people and developers must work together daily throughout the project* (Principle 4).
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation* (Principle 6).

Early and continuous delivery, as specified by Principle 1, is achieved by adopting an iterative, incremental project management approach, such as *Scrum* (Schwaber & Beedle, 2002). In Scrum,

iterations are called *Sprints* and usually last 2-4 weeks. During Sprints, functionalities are built to the product. For representing cumulative, remaining work, *burn down charts* are used. At the end of each Sprint, results of development activity are presented and evaluated by management and customers.

High-level requirements are defined at the beginning of the project and collected in what is called the *Product Backlog*. At the beginning of each iteration, customers and development team discuss which of these requirements must be implemented in the current Sprint, prioritizing and eventually modifying requirements using knowledge generated within each iteration as a guide. As it can be evinced, requirements are not stated once and for all, but they are subjected to change throughout the project, as according to Principle 2. At the beginning of each Sprint, goals and functionalities are chosen from the Product Backlog and included in the *Sprint Backlog*. *User Stories* can be used to represent requirements and features from the point of view of the customer in an effective way (Robinson & Sharp, 2010).

In order to fully accomplish his role, a representative of the customer should physically sit with the team to "answer questions, resolve disputes, and set small-scale priorities" (Beck & Andres, 2004). The customer, as suggested by Principle 4, would be collaborative, representative, authorized, committed and knowledgeable for the development team by sharing the same working space, being herself a full-time member of the team. As all members of the team, she should work daily throughout the project in a close way, while communicating frequently with other members. Close collaboration is thought as the best way to build good rapport between team members: the mutual understanding of each other's role, abilities, tasks and issues is achieved by increasing informal communication and easing initiating contact. As a result, team cohesion is fostered.

Principle 6 states that face-to-face conversation should be the preferable way for communicating within the development team (which, as stated before, also comprises a customer representative). In fact, frequent, informal communication is an essential element of Agile methodologies, being the preferable method for monitoring the state of a project and for building team cohesion (Ramesh, et al., 2006). Face-to-face communication has also the merit of reducing the inherent ambiguity of text-based, asynchronous communication (Korkala, et al., 2009; Damian & Zowghi 2002).

CUSTOMER-DEVELOPER COLLABORATION IN DISTRIBUTED AGILE SETTINGS

In Distributed Agile Development, the three distances related to Distributed Software Development (i.e. Geographical, Temporal, Socio-cultural) can seriously affect the principles and practices that have been listed in the previous section (Holmström et al., 2006).

Geographical distance between customer and developers implies that co-located work (Principle 4) is not easily feasible:

- Even when it is possible for a customer representative to move and work on-site with the development team, it has been noticed that the customer herself could in turn lose connection with her own environment (Dullemond et al., 2009).
- In a distributed scenario, frequent communication becomes sensibly harder and the possibility of having informal communication substantially disappears. This can have various consequences: the up-to-date knowledge of a project's status and progresses relies mostly on informal exchanges (Principle 1 and 2) and therefore the emergence of problems may not be easily detected as in a co-located environment.
- Scheduling meeting between customers and the development team is possible, although attending to them might be costly due to long travels (P2).
- Informal, face-to-face communication could be replaced by video conferencing, although some effort could be necessary in order to initiate communication without the implicit knowledge of other people's current status that co-location allows (Ågerfalk, 2005) (P6).

Temporal distance is especially critical to coordination processes:

- Synchronous, face-to-face communication (even via video conferencing) (P6) is restricted by differences in availability of the counterparts.
- Information about reciprocal availability is not necessarily well known, predictable or even easy to infer without some form of previous agreement.

Socio-cultural issues are related to the lack of team cohesion and vision that separation between customer and developers can imply (P4):

- Agile methodologies privilege individuals over processes (Fowler & Highsmith, 2001), but it can get considerably harder to foster interaction when knowledge about people, their role and current tasks is not easily available.
- Lack of control over the development process makes difficult for customers to have feedback about the project and to check its adherence to requirements, therefore leading to lack of trust (P1 and P2).
- The absence of an efficient customer-developer relationship caused by lack of trust can affect the whole Agile development process, possibly leading to radicalization of procedures, information hiding and disengagement (Korkala, et al., 2009).

As a result of this analysis, we can identify four targets that a system supporting distributed, Agile customer-developer collaboration must achieve:

- Target A: *Tracking of distributed process iterations*. This target impacts control activities (e.g. obtaining a shared knowledge of the project's status).
- Target B: *Requirements definition and presentation*. This target impacts communication activities (e.g. formal meetings for requirements definition) and coordination activities (e.g. assigning tasks to developers and tracking their issues).
- Target C: *Close collaboration*. This target impacts communication activities (e.g. daily, informal communication) and coordination activities (e.g. identifying the right collaborators for initiating contextual and information-rich conversations).
- Target D: *Synchronous, face-to-face communication*, which is the preferred modality for formal and informal communication activities.

In Table 1, each target is associated with a correlated Agile Principle and with the distances that it must overcome in a distributed setting.

Agile Principle	**Target**	**DSD Distance**		
		GD	***TD***	***SCD***
Principle 1	Tracking of distributed process iterations	x		x
Principle 2	Requirements definition and presentation	x		x
Principle 4	Close collaboration	x		x
Principle 6	Synchronous, face-to-face communication	x	x	

Table 1. Targets

THE SCOAP FRAMEWORK

Features

The four targets we have identified in the previous section served as a basis for designing features and composition of the SCoAP framework:

- Target A requires a support for sharing information about the status of a project. Coherently with Agile principles, a project can be organized in iterations or Sprints, that define the "pace" of the project and help tracking its progress.
- Target B requires a support for defining and organizing project requirements: User Stories can be used for defining functionalities, while Tasks individuate smaller units of work that can be assigned to specific developers. Requirements are normally discussed in formal meetings, which, in a distributed setting, may have to be scheduled and attended in a virtual environment.
- Target C requires collaboration between customers and developers to be as information-rich as in co-located settings: User Stories and Tasks can be organized within a structure called *Taskboard*, which presents a spatial distribution of Tasks based on their status, thus delivering an overall picture of the development progress and a focal point for coordination and communication (Robinson & Sharp, 2010).
- Target D requires the use of a rich communication channel: video conferencing can reduce ambiguity about the message and uncertainty about its interpretation (Korkala, et al., 2009). However, communicating through a rich channel could not be enough: users must be aware of other people's status, role and current tasks in order to contextualize their communicative activities.

Layers

The SCoAP framework is composed of three layers: *User Definition Layer*, *Project Layer* and *Communication Layer*. Each layer of the framework is strictly interleaved with the others, either as a source of information or as a tool for completing certain activities: put in other words, the former two layers are sources of contextual information that is to be exploited in formal and informal communication activities provided by the latter.

User Definition Layer

The User Definition Layer (UDL) collects and presents information about users. People who have the necessary permissions (i.e. other developers, managers and customers working in the same project) can access information such as identity, role and current activities of others. Information can be either specified by a human (i.e. the actual user or a system administrator) or it can be automatically defined by the system on the basis of the actual status of the specific user whose information has to be updated.

Specifically, a user **U** is represented as a tuple of six elements (**N**, **Pic**, **Loc**, **R**, **AT**, **Av**), where:

- **N** is the name of the user: it is represented as a string and it is user-specified.
- **Pic** is the picture of the user: it is represented as an image and it is user-specified.
- **Loc** is the current geographical location of the user: it is represented as a string and it is user-specified.
- **R** is the role of the user: it is represented as a string and it is user-specified. Users' role can either be customers or members of a development team. Other stakeholders, such as testers and project managers, can also be provided with a user specification if their contribution is needed.
- **AT** is the set of active (i.e. not yet completed) tasks that have been assigned to the user: it is represented as a list, whose elements are automatically derived from information contained in the Project Layer (see below).
- **Av** is the current availability of the user, that can be one within this set of values: *Available*, *In a call*, *Idle*, *Offline*. The availability is automatically defined by the system on the basis of users' presence within the video conferencing tool (see section *Implementation* for details about the video conferencing tool).

Project Layer
The Project Layer (PL) collects and presents information about definition, status and progress of a project. Such information is modeled after the principles of the Scrum methodology: projects are organized in Sprints, User Stories and Tasks. Elements are generally user-specified and user-updated, although, as in UDL, some information can be automatically defined by the system.

A Sprint **S** is represented as a tuple of four elements (**SN**, **SSt**, **BD**, **ED**), where:

- **SN** is the name of the Sprint.
- **SSt** is the current state of the Sprint, that can be one within this set of values: *Programmed*, *Current*, *Finished*.
- **BD** is the begin date of the Sprint.
- **ED** is the end date of the Sprint.

A User Story **US** is represented as a tuple of two elements (**USN**, **USS**), where:

- **USN** is the name of the User Story.
- **USS** is the name of Sprint in which the User Story has been allocated.

A Task **T** is represented as a tuple of six elements (**TN**, **TUS**, **TU**, **TUAv**, **TDu**, **TSt**), where:

- **TN** is the name of the Task.
- **TUS** is the name of the User Story in which the Task has been allocated.
- **TU** is the name of the user to which the Task has been assigned.
- **TUAv** is the current availability of the user to whom a specific Task is assigned, that can be one within this set of values: *Available*, *In a call*, *Idle*, *Offline*. Availability information is automatically defined by the system on the basis of the Task assignee's presence within the video conferencing tool.
- **TDu** is the expected duration of the task, represented as hours;
- **TSt** is the current state of the Task, represented as a set of values (*Assigned*, *Ongoing*, *Completed*).

Communication Layer
The Communication Layer (CL) is a set of practices and tools that supports rich and flexible communication between customers and developers. We identified three possible communication modalities that could support the our prefixed level of flexibility:

- *One-to-one*, in which the customer communicates with a single developer. This communication modality is useful for direct communication about specific issues, i.e. a developer asks some clarification about a requirement to the customer, or the customer asks questions about the work of a developer on a specific Task.
- *One-to-many*, in which the customer communicates with two or more developers, but not with the whole development team. This situation is quite similar to the previous one, but it can be centered on a broader subject, such as the development activity regarding a whole User Story, that can involve several developers.
- *One-to-all*, in which the customer communicates with the whole development team. This modality can be applied to Sprint reviews, where the whole team and the customer discuss about requirements to be implemented in the next Sprint or about the results of the last iteration.

Implementation

We developed a prototype of a SCoAP framework implementation using *Microsoft SharePoint Server 2010*[i] for building the dynamic website and *Microsoft Lync Server 2010*[ii] as the video conferencing platform. The dynamic website exposes most of the functionalities of the framework, while video conferencing has been chosen as the preferred communication channel for real time communication between remote sites.

Each user has an identity for logging in the website and in the Lync client application. The website features client-level integration with the Lync application, which allows users to operate with the Lync client directly from the website through specific widgets. The Lync client, in turn, updates availability information about users that is presented on the website.

A SharePoint Server farm handles business data belonging to UDL and PL, while the CL business data are handled by a Lync Server Farm. The two farms belong to the same Active Directory domain, therefore they share user accounts and identities, allowing Single Sign-On between the two services: in fact, a coherent authentication modality, within a unifying directory service, has been shown to be a preferable choice in a collaborative environment that offers multiple tools (Sinha et al., 2006; Ardissono et al., 2011; Ardissono & Bosio, 2012).

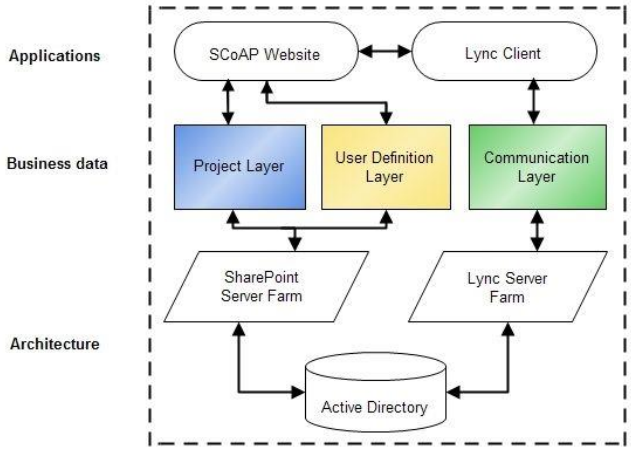

Figure 1. Technical details

The UDL application level is presented as a dynamic web page that organizes and presents the properties for each participant to the project, allowing users to visualize and edit such properties if they possess the necessary permissions. Users can also initiate direct communication with others, according to their availability status, directly from their user profile thanks to a widget which is integrated with the conferencing client. Users are divided in teams: such division is based on their geographical location and on the project(s) they are working at. Knowing the location of a team is critical in a distributed scenario, especially when it should be possible to contact the whole team at once through a virtual meeting room (see below). Accessing the virtual meeting room with the team is possible by clicking the corresponding banner on the UDL page.

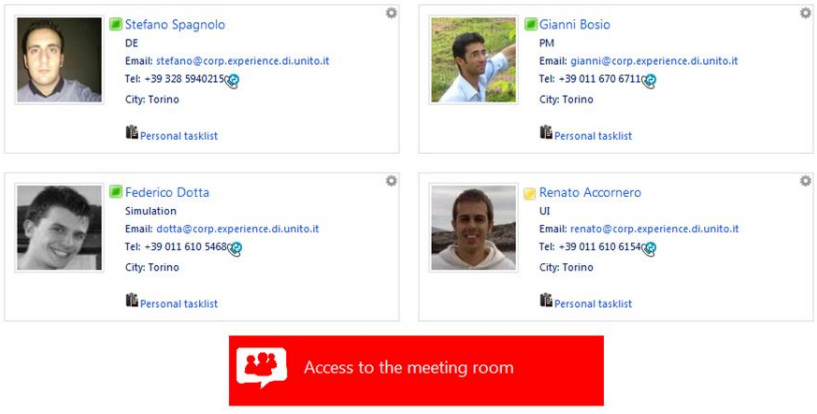

Figure 2. User Definition Layer implementation

The PL application level is presented as a collection of dynamic web pages. The first page presents a view of the overall status of the project: it lists all the Sprints that have been defined and their properties, and a graphical indicator of the project's status as a whole is also displayed.

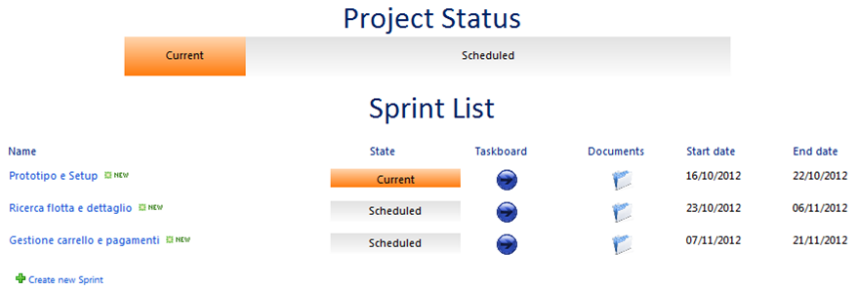

Figure 3. Project Layer: Project status and Sprint list

Users can access detailed view about each Sprint by clicking on their names: such details are presented in a specific web page modeled with the form of a Taskboard. From the Taskboard, users can create and visualize User Stories and Tasks.

Tasks are represented as "sticky notes" on the board and organized into three columns on the basis of their current status (Planned, Ongoing, Completed). Each Task note has a different color on the basis of the User Story to which it has been allocated.

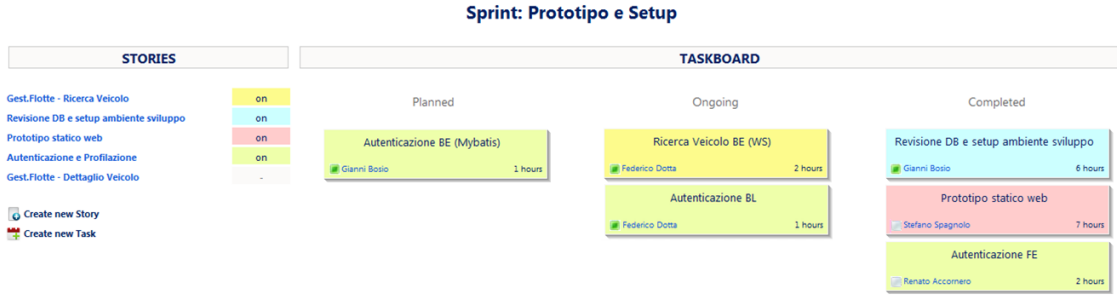

Figure 4. Project Layer: Taskboard

For each task, information about the availability of its assignee is presented, so that other users can initiate direct communication when possible just by clicking on the availability widget itself.

Figure 5. Project Layer: availability widget

A burn down chart is associated to each Taskboard in order to graphically present the progress of the Sprint in terms of remaining work and to track its adherence to the planned timeline. A *user chart* depicts the percentage of completed tasks for each member of the development team.

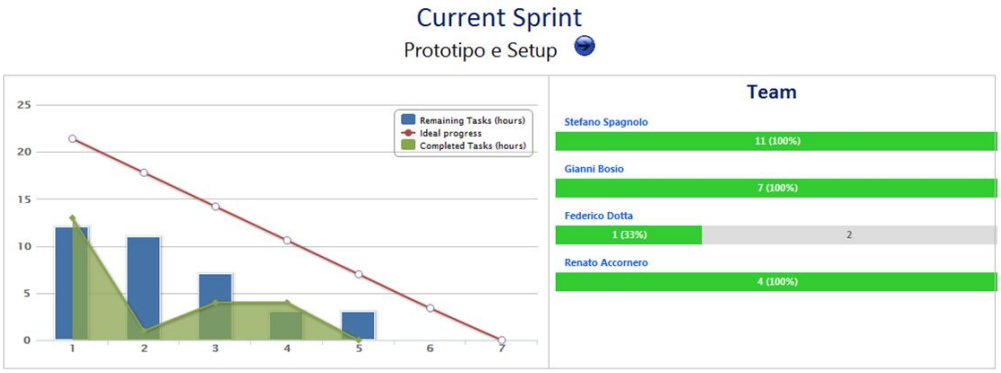

Figure 6. Project Layer: Burn down chart and user chart

The first two modalities of the CL (one-to-one and one-to-many) are supported via direct video calls between the customer and the developers: as we have seen above, direct calls can be initiated from within users' and project's pages thanks to the availability indicators. However, a simple call (or multi-call) could not be enough for the third modality (one-to-all), as it may involve too many people and therefore it could be quite cumbersome to handle.

To handle such a scenario, a proper meeting room, equipped with a specific camera and a video projector, should be prepared. We have made this room easily accessible by means of a direct link on the User Definition Layer web page (see Figure 2): links to meeting rooms can be created using the video conferencing application and they can be successively added to the web page.

The meeting room is meant to coincide with the development team's office, in order to support formal meeting as well as informal access and communication with the developers' environment. This aspect could pose some problems when the development team itself is dispersed in several locations (a situation which could be common in DSD settings) and several meeting rooms have to be prepared; this is the reason why the UDL page adapts by dividing developers in teams on the basis of their geographical position, and by permitting to attach a proper link for accessing the corresponding meeting room.

Figure 7. Communication Layer: One-to-all video call

RELATED WORK

Several tools exist that support Agile software development; all of these tools present features for handling project management, tasks management, Backlog management and requirements tracking:

- The *Agilefant* tool (Vähäniitty & Rautiainen, 2008) supports iterative and incremental development work, with particular attention to linking daily work items with business level goals (business-alignment). It expands the notion of Backlog management introducing Epics, which differ from regular Backlog items in terms of dimension (they are much bigger) and nature (they are not necessarily part of software requirements).
- *Rally Enterprise Edition*[iii] is a commercial agile software development life cycle management software. It contains functionalities for project management and requirements tracking that follow Scrum philosophy (projects consist of releases and iterations). It features User Stories and tasks, with the possibility to add "widgets" that can show relevant information (e.g., the list of tasks assigned to the current user or an iteration burn down chart).
- *ScrumWorks Pro*[iv] is another commercial agile software development management tool which has been designed to be used within Scrum projects: it features a web-based interface which provides a list of all Backlog items and tasks for a selected Sprint. Backlog items are used for planning high-level work activities and are then assigned to Sprints; work activities can then be planned at a lower level by creating tasks which are related to specific Backlog items.

Although these tools can be very powerful in supporting work within their field of appliance, they usually lack features that are at the basis of SCoAP, i.e. those that permit the customers' involvement within development activity and the tight integration of development activities with continuous, rich communication (targets C and D of our analysis). As a consequence, we think that such tools can represent a powerful addendum to an Agile project, but they do not holistically address the problem as we meant to do.

We have also realized that the scientific literature about the problem of handling customer – developer collaboration in a distributed Agile context is still quite scarce. Therefore, we broadened our analysis on the various issues of distributed Agile development and then focused on themes related to communication and collaboration.

Cottmeyer (2008) stresses the importance of collaboration tools in distributed Agile development, especially those dedicated to the measurement of metrics such as team velocity, individual velocity, project burn down, major milestones and Backlog items: these are all items that the Project Layer of our framework fully encompasses, as they are key points in establishing trusting relationships between development teams and customers.

Layman et al. (2006) present four conjectures concerning the success factors for globally-distributed Agile teams: the importance of a well-defined customer authority for effective decision and requirements statement, the communication conduit that a physically located member of another team can provide, the positive impact of prompt responses to asynchronous queries and the improved process control and plan effectiveness that continuous access to process and product information can provide. While we did not focus on the second and third aspect (mainly because they are not directly related with themes covered by our framework), we definitely considered as fundamental the other two: in particular, our framework requires customer involvement to be well-defined and continuous during the project; moreover, we foster process and product information sharing through our Project Layer.

Dorairaj et al. (2011) present a Grounded Theory study that explores communication challenges in distributed agile environments and lists strategies that could be adopted to overcome them. Challenges were identified as lack of communication tools, time zones, language barriers and lack of teamwork; in order to overcome these challenges, the authors propose to reduce time zones, leverage communication tools and techniques such as video conferencing, addressing language barriers and increasing formal and informal communication. Our study is coherent with this analysis, as we propose a framework and a tool for collaboration and communication within a distributed, Agile setting.

Paasivaara et al. (2009) present a single case study of a large distributed product program using an agile process. In this study they explain how daily Scrum meetings increased transparency between sites, allowing participants to obtain a good overview of what was happening in the project, and finally enhanced communication across sites. Improvements were found in communication, trust, motivation and product quality: such benefits also led to more one-to-one communication between the sites than before. We think that our framework can successfully promote communication, trust and motivation between customers and developers, as it allows people to control the amount of information they share with the others and, at the same time, to access all valuable information about the other participants for the sake of the development project.

Korkala et al. (2009) present a single case study of a large globally distributed organization, focused on communication with customers during requirements engineering and software implementation. In their study, they found that communication happened mostly through telephone or asynchronous tools (email, wiki) and that developers where largely excluded from direct communication with customers, mainly due to organizational politics that restricted information sharing: this resulted in largely insufficient communication. We recognize that restrictive policies about information sharing can definitely hamper the results of an Agile, distributed project: therefore, we promote the use of an inclusive, bottom-up framework for handling information disclosure.

Persson et al. (2012) analyze the importance of formal and informal control strategies within distributed Agile projects. They propose a tool named *Comapping* that, similarly to our framework, can be used for Backlog management and for the visualization of tasks and task status: as such items represent a set of shared commitments to the team, they can also be seen as formal control elements, while they were not perceived as impediments to agile practices; informal control practices can also be supported via real-time communication.

CONCLUSION

In this paper we have presented SCoAP, a framework that supports for communication, coordination and control activities between customers and developers in a distributed, Agile software development setting. In order to define the framework, we have identified several targets based on Agile principles applied to DSD issues. The resulting framework collects and presents information about users that is useful for scenarios where it is impossible to acquire knowledge from co-location and informal communication. Information about definition, status and progress of Agile projects is also collected and presented, in order to enable customers as well as developers to acquire an overall picture of the progress of the development process and to serve as focal point for collaboration and communication activities. Real time communication between remote sites is obtained through video conferencing: we defined three modalities of video communication that support information rich communication, suitable for both formal and informal activities.

This study analyzes existing literature regarding both DSD and Agile methodologies and all its deductions and findings are literature supported. However, we are looking forward to validate the results of this study by investigating the actual usage of the framework in a real scenario. Future work will include a case study regarding the adoption of SCoAP: such study could also allow us to expand the areas covered by our research, including the impact of socio-cultural differences related to languages and practices and the privacy concerns that a pervasive communication infrastructure like the one we have conceived could have on its users.

REFERENCES

Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., & Conchúir, E. O. (2005). A framework for considering opportunities and threats in distributed software development. In *International Workshop on Distributed Software Development* (pp. 47-61).

Allen, T. J., & Fustfeld, A. R. (1975). Research laboratory architecture and the structuring of communications. *R&D Management, 5*(2), 153-164.

Ardissono, L., Bosio, G., Goy, A., Petrone, G., Segnan, M., & Torretta, F. (2011). Collaboration Support for Activity Management in a Personal Cloud Environment. *International Journal of Distributed Systems and Technologies (IJDST)*, 2(4), 30-43.

Ardissono, L., & Bosio, G. (2012). Context-dependent awareness support in open collaboration environments. *User Modeling and User-Adapted Interaction,22*(3), 223-254.

Beck, K., & Andres, C. (2004). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

Bergadano, F., & Bosio, G. (2013). A contextual framework supporting collaboration between customers and developers in distributed, agile software development. In *IADIS Int. Conf. on Collaborative Technologies 2013 (CT 2013)* (pp. 35-42). IADIS Press.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer, 21*(5), 61-72.

Carmel, E. (1999). *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR.

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *Software, IEEE, 18*(2), 22-29.

Carmel, E., & Abbott, P. (2006). Configurations of global software development: offshore versus nearshore. In *Proceedings of the 2006 international workshop on Global software development for the practitioner* (pp. 3-7). ACM.

Carmel, E., & Abbott, P. (2007). Why 'nearshore' means that distance matters. *Communications of the ACM, 50*(10), 40-46.

Conchuir, E. O., Holmstrom, H., Agerfalk, J., & Fitzgerald, B. (2006). Exploring the assumed benefits of global software development. In *Global Software Engineering, 2006. ICGSE'06. International Conference on* (pp. 159-168). IEEE.

Cottmeyer, M. (2008). The good and bad of Agile offshore development. In Agile, 2008. AGILE'08. Conference (pp. 362-367). IEEE.

Damian, D. E., & Zowghi, D. (2002). The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on* (pp. 319-328). IEEE.

Damian, D., & Moitra, D. (2006). Guest Editors' Introduction: Global Software Development: How Far Have We Come?. *Software, IEEE, 23*(5), 17-19.

Dorairaj, S., Noble, J., & Malik, P. (2011). Effective communication in distributed Agile software development teams. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 102-116). Springer Berlin Heidelberg.

Dullemond, K., van Gameren, B., & van Solingen, R. (2009). How technological support can enable advantages of agile software development in a GSE setting. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on* (pp. 143-152). IEEE.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*,*9*(8), 28-35.

Glass, R. L. (2001). Extreme Programming: The Good, the Bad, and the Bottom Line. *IEEE Software*, *18*(6), 112-111.

Herbsleb, J. D., & Moitra, D. (2001). Global software development. *Software, IEEE*, *18*(2), 16-20.

Herbsleb, J. D., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, *29*(6), 481-494.

Holmström, H., Fitzgerald, B., Ågerfalk, P. J., & Conchúir, E. Ó. (2006). Agile practices reduce distance in global software development. *Information Systems Management*, *23*(3), 7-18.

Korkala, M., Pikkarainen, M., & Conboy, K. (2009). Distributed agile development: A case study of customer communication challenges. In *Agile Processes in Software Engineering and Extreme Programming* (pp. 161-167). Springer Berlin Heidelberg.

Layman, L., Williams, L., Damian, D., & Bures, H. (2006). Essential communication practices for Extreme Programming in a global software development team. Information and software technology, 48(9), 781-794.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using scrum in distributed agile development: A multiple case study. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on* (pp. 195-204). IEEE.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on* (pp. 308-313). IEEE.

Persson, J. S., Mathiassen, L. and Aaen, I. (2012), Agile distributed software development: enacting control through media and context. *Information Systems Journal*, 22: 411–433.

Prikladnicki, R., Audy, J. L. N., & Evaristo, J. R. (2003). Distributed Software Development: Toward an Understanding of the Relationship Between Project Team, Users and Customers. In *ICEIS (3)* (pp. 417-423).

Ramesh, B., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile?. *Communications of the ACM*, *49*(10), 41-46.

Robinson, H., & Sharp, H. (2010). Collaboration, Communication and Co-ordination in Agile Software Development Practice. In *Collaborative Software Engineering* (pp. 93-108). Springer Berlin Heidelberg.

Saiedian, H., & Dale, R. (2000). Requirements engineering: making the connection between the software developer and customer. *Information and Software Technology*, *42*(6), 419-428.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*(Vol. 1). Upper Saddle River: Prentice Hall.

Sinha, V., Sengupta, B., & Chandra, S. (2006). Enabling collaboration in distributed requirements management. *Software, IEEE*, *23*(5), 52-61.

Tseng, M. M., & Du, X. (1998). Design by customers for mass customization products. *CIRP Annals-Manufacturing Technology*, *47*(1), 103-106.

Vähäniitty, J., & Rautiainen, K. T. (2008). Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. In *Proceedings of the 1st international workshop on Software development governance* (pp. 25-28). ACM.

Zhu, L., Barricelli, B. R., & Iacob, C. (2011). A Meta-Design Model for Creative Distributed Collaborative Design. *International Journal of Distributed Systems and Technologies (IJDST)*, 2(4), 1-16.

[i] http://office.microsoft.com/sharepoint/
[ii] http://office.microsoft.com/lync/
[iii] http://www.rallydev.com/
[iv] http://www.collab.net/products/scrumworks